IPB: An Internet Protocol Benchmark Using Simulated Traffic

Bruce A. Mah
bmah@CA.Sandia.GOV
Sandia National Laboratories / California
P.O. Box 969, M.S. 9011
Livermore, CA 94551-0969

Peter Sholander, Luis Martinez, Lawrence Tolendino
{peshola,lgmarti,lftolen}@Sandia.GOV
Sandia National Laboratories / New Mexico
P.O. Box 5800
Albuquerque, NM 87185

Abstract

A common shortcoming of many network measurement programs is that their metrics are not expressed in terms directly applicable to applications and users. To address this problem, we have developed the IP Benchmark (IPB), a benchmark program that synthetically generates traffic to simulate the network activity of common Internet applications. It measures the performance experienced by the simulated applications, and expresses that performance in terms of application-specific metrics. After presenting some details of the design and implementation of IPB, we show how we use it to measure HTTP performance along various long-delay, lossy ATM links, such as might be found in satellite or wireless ATM environments. These links themselves are simulated by using a BBN Long link Emulator, which adds delays and errors to local-area OC-3 links. Finally, we present some preliminary results.

1 Introduction

A challenge in evaluating the performance of computer networks is the ability to do so in a manner meaningful to applications and users. While commonly-used microbenchmarks such as `ttcp` [16] give some useful information such as raw throughput and round-trip time, they frequently do not provide the type of data meaningful to users. Many users' concerns revolve around metrics such as the response time while Web browsing, keystroke echo time during remote login sessions, and the quality of network video.

We have developed an *IP Benchmark* (IPB), which uses synthetic, simulated network traffic to measure the performance across an IP network or internetwork, with respect to the requirements of different Internet applications. Currently, it incorporates simulations of the telnet remote login service and the Hypertext Transfer Protocol (HTTP) used by World Wide Web applications. Both simulations are based on empirical models derived from traffic traces, and both rely on the functionality of the actual TCP/IP protocol stacks in the hosts running the benchmark. IPB supports running a mix of various applications simultaneously. IPB's network performance measurements are expressed in terms of application-specific metrics, such as the response time for telnet connections or the time to request and receive a Web page.

We have used IPB to measure the performance of HTTP over lossy links with various delays, such as might be found in a satellite or indoor wireless network. Our experimental setup uses a BBN Long Link Emulator (LLE) to provide an environment with tunable delay and error characteristics. Although the LLE is an OC-3 SONET device, we run IP over ATM across the SONET links.

In Section 2, we describe our benchmark software, with an emphasis on its simulation of HTTP traffic based on an empirically-derived model. Section 3 briefly outlines the network testbed we are using to investigate satellite and wireless links, through network emulation. In Section 4 we present the results of our initial experiments with these various components. Finally, we present a summary and some directions for future work in Section 5.

2 IP Benchmark

To investigate network performance with respect to real-world Internet applications, we have constructed a traffic generation and measurement program, rather unimaginatively titled the *IP Benchmark* (IPB). IPB can be run between any two IP hosts to test the performance of various network applications sending data over the path between those hosts. Like actual network applications, IPB uses the TCP, UDP, and IP implementations on the two endhosts, thus allowing measurement studies made using IPB to capture the effects of different protocol implementations and operating systems.

IPB is written in C++, and consists of two executable programs, to be run on each end of a path or link to be tested. A client application `ipb` initiates connections and sends data on them, to simulate the activity of a number of users running some number of applications. The arrival of new user sessions is generated according to a set of Poisson processes (one per application type), while the behavior of each conversation is controlled by application-specific traffic models. A server application ipbd accepts connections from the client. It responds as necessary to data sent by the client, for example, in the form of keystroke echoes for a telnet remote login session. Both programs log the performance characteristics of the simulated network applications.

We note that ipb and ipbd only mimic the actions of various Internet applications; they do not contain or rely on the actual programs themselves. However, both ipb and ipbd send data over the network using the endhosts' IP stacks, using standard sockets. This approach allows us to observe any artifacts caused by the protocol implementations.

To avoid overheads caused by process context switches, both ipb and ipbd are implemented as single processes, and take it upon themselves to manage the state for a potentially large number of active sessions. This approach had the noticeable effect of increasing the implementation complexity over a design using a process per conversation.

ipb and ipbd both share a similar structure, patterned somewhat after an event-driven network simulator. In fact, the implementations of the traffic models were adapted from those in the INSANE IP-over-ATM network simulator [6, 7]. In ipb and ipbd, there are two types of events that can be handled by the simulation code. Timer events control the arrival of external events, such as the entrance of new users into the system or the behavior of current users. Network I/O events signal the readiness of sockets for reading or writing or (in the case of ipbd) the arrival of a new connection. Generic pseudo-code for the main event loops of ipb and ipbd is shown in Figure 1.

```
Initialize();
event = FindNextExternalEvent();
while (true) {
     // Select to find ready sockets,
     // with timeout based on external
     // event from queue.
     n = select();
     if (n == 0) {
          // No I/O, timer event was next.
          // Dispatch timer event, based
          // on what application posted
          // it. The handler may queue up
          // new external events.
          Handle(event);
          Dequeue(event);
          event = FindNextExternalEvent();
     }
     else {
          // Dispatch I/O event, based
          // on what socket caused it.
          // These handlers may also queue
          // new external events.
          sock = FindSocket();
          Handle(sock);
     }
}
```

FIGURE 1. Pseudo-Code of ipb and ipbd Event Loops.

In the remainder of this section, we present some details about IPB's simulation of HTTP network traffic. IPB also incorporates a simulation of the telnet remote login service [13], based on the empirical model of [4].

2.1 Model of HTTP Traffic

To simulate network activity caused by World Wide Web applications, IPB uses a model of network traffic generated by applications using the Hypertext Transfer Protocol (HTTP) [3, 5]. HTTP is a request-response protocol used to transfer files between Web clients and servers; for reliability, HTTP relies on the services of the TCP transport protocol [12, 15]. Web documents, or "pages", are composed of a series of these files, each of which are requested and transferred individually. The traffic model used by IPB is defined by a small number of probability distributions, computed by gathering and analyzing traffic traces of World Wide Web conversations. The different quantities modeled are summarized in Table 1. Further details on this model and its derivation can be found in [8].

2.2 Simulated HTTP Client

ipb contains code to simulate the actions of a number of HTTP client processes, all running independently. Each simulated HTTP retrieval begins with a request message (characterized by the *request length* distribution), sent by TCP to the simulated HTTP server. ipb matches each reply with its corresponding request message, and records the round-trip time (RTT) for each retrieval. Web pages are represented by a number of successive retrievals (drawn from the *document size* distribution).

To simulate the use of the "Keep Alive" extension in HTTP/1.0 and HTTP/1.1, each client session keeps only one TCP connection open at a time, and reuses it for multiple retrievals[1]. Moreover, ipb "pipelines" the requests for a given Web page, so that all requests after the first on the page are made at once, without waiting for their replies. This behavior simulates a Web browser requesting and

1. Alternatives to "persistent-connection HTTP" include the use of individual TCP connections to transfer files in series, and the use of multiple connections in parallel, first seen in [10].

Quantity	Description
request length	HTTP request length. Request sizes are bi-modal, with about 70% of requests around 250 bytes long and about 15% around 1 KB.
reply length	HTTP reply length. Replies have a heavy-tailed distribution, with a median of 1.5–2.0 KB and a mean of 8–10 KB. Can be approximated with a Pareto distribution.
document size	Number of files per document. Although the mean document consists of 2.8–3.2 files, more than half of all documents consist of only one file.
think time	Time between retrieval of two successive documents, with a median of 15 seconds.
consecutive document retrievals	Number of consecutive documents retrieved from a given Web server. The mean visit to a server retrieved four documents, although the median visited retrieved two.
server selection	Chooses a Web server to visit. Not used by IPB.

TABLE 1. HTTP Traffic Parameters Modeled.

receiving HyperText Markup Language (HTML) text [2] for a page, determining the set of "secondary" files needed (such as in-line images), and then requesting them all as a batch. In addition to the response time for each file retrieved, `ipb` records the total response time for each Web page. We note that because of the pipelining of multiple HTTP requests, the requests and replies for the files making up a Web page may overlap. Thus, the response time for a Web page will frequently be less than the sum of the response times for its component files. Figure 2 illustrates the relationship between HTTP files and Web pages.

Between documents, the client pauses for a random interval of time to simulate the user's reading of the newly-retrieved page and determining her next action. This length of time is described using the *think time* probability distribution.

Each simulated client requests a total number of Web pages drawn from the *consecutive document retrievals* component of the HTTP model. After this point, the client ceases operations, on the assumption that the associated user has shifted her attention to a new Web server or has closed her Web browser. New clients enter the system according to a Poisson process, with an average interarrival time specified from the `ipb` command line.

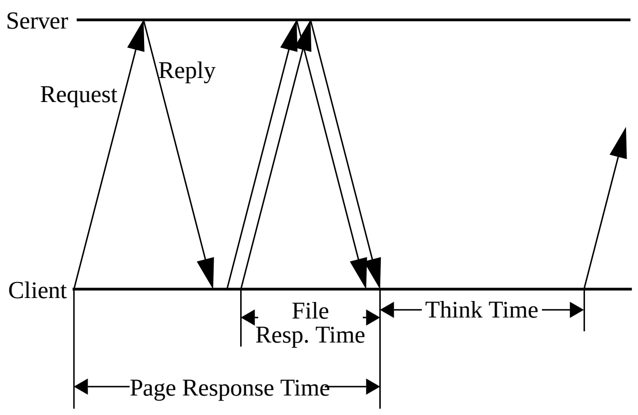

FIGURE 2. HTTP File Transfers. Diagonal arrows represent requests (upward) and replies (downward), as time proceeds to the right. Pictured are the interactions needed to retrieve a single Web page comprised of three files.

2.3 Simulated HTTP Server

The functionality of the simulated HTTP server in `ipbd` is considerably simpler than that of the client. For each currently-established connection, `ipbd` receives HTTP requests and returns responses whose sizes are drawn from the *reply length* distribution. Each reply is sent on the same TCP connection that carried its corresponding request.

The implementation of the HTTP server, however, is complicated somewhat by our original desire to use a single process for the benchmark server. To simulate the actions of multiple, potentially-blocking processes by a single process requires some effort. We chose an approach in which `ipbd` composes outgoing replies and attempts to send them as non-blocking sends. If transmission of an HTTP response cannot be completed in full, `ipbd` records the remaining portion of the response. A write of the remaining data is reattempted once the TCP socket is once again ready for writing; in the meantime the `ipbd` process is free to deal with other connections.

Because the simulated HTTP data transported is merely a test pattern (arbitrary bytes with NULL/zero bytes used to delimit responses) it is fairly easy to reconstruct remaining partial responses. For each TCP connection used for HTTP, `ipbd` only needs a record of the number of bytes remaining in the current partial response (if any) and a list of the sizes of any pending responses queued up behind it.

We note that there are a number of approaches to structuring Internet server programs. Many modern Web servers, such as Apache [1], rely on a pool of processes to handle multiple pending HTTP requests. With this approach, a requests that gets blocked due to TCP congestion/flow control, disk I/O, or other processing delays will have minimal impact on other requests. In simpler, single-process servers, a blocked request causes all other pending requests to be queued behind it. Others, such as the Squid HTTP proxy/cache [14], rely on non-blocking I/O in essentially the same way as `ipbd`.

3 An Emulated Wireless/Satellite Environment

We have used IPB to examine the performance of HTTP applications across various types of SONET links. In order to investigate a space of different types of links (characterized by their delay and loss properties), we used a BBN *Long Link Emulator* (LLE) [9] to simulate a wireless or satellite link in a controlled setting. The LLE is a device that can induce delays and errors in OC-3 links, to simulate the effects of long transmission links. It causes delays up to approximately 206 ms (in each direction) per link by buffering data in its on-board memory[2]. Errors can be created through the use of hardware error generators, which can be set to produce errors of specified frequency and duration. The LLE operates on data at a bit level, so it is oblivious to SONET framing or any higher-layer protocols. It can forward data at the full OC-3 line rate of 155 Mbps, on up to five bidirectional links, with both directions of each link controlled independently.

The LLE's operation is programmable. It contains an on-board SPARC processor, RAM, and a small hard disk. The LLE runs a suitably-modified version of SunOS, and can appear as an IP host on a network through a standard Ethernet interface. Software included with the unit allows a variety of methods to change the delay and error properties, either manually or under a program's control. However, the SPARC processor is not on the data forwarding path; delays and errors are created entirely by the LLE hardware.

The LLE's on-board error generator is capable of producing either various types of errors at random, such as bit inversions, "all zeros", or random noise. It uses a pseudo-random number generator, driven by a hardware clock, to control the timing of errors. The error generators can create random errors with a byte corruption probability of 2^{-37} to 2^{-6}. Errors occur in fixed-sized bursts of between one and eight bits.

2. The propagation time to a satellite in geosynchronous orbit is approximately 125 ms, for an up-and-down propagation time of approximately 250 ms, slightly longer than the 206 ms capability of a single link buffer on the LLE. Multiple links can be cascaded to produce longer delays.

For some scenarios, more sophisticated controls over errors will be required. The hardware has only two parameters for describing random errors (the probability of byte corruption and the fixed length of error bursts, in bits). These two parameters, by themselves, are insufficient to describe more complex loss patterns. For example, an empirical study of an indoor wireless network showed error-free periods and error bursts whose lengths were described by two empirical probability distributions or analytically approximated by a set of five distributions [11]. Simulating such loss patterns requires a more flexible characterization of errors.

In some scenarios involving mobility or low-earth-orbit (LEO) satellites, it may be useful to modify a link's delay while data is being transmitted. However, the LLE delay hardware causes "glitches" in a link whenever its delay is changed. Modifying the link delay during active operations is not recommended, as the "glitch" can last as long as the new link delay.

The remainder of our network testbed consists of two Sun Sparcstation 10s running SunOS 4.1.3 with FORE ASB-200 ATM NICs. The ATM NICs were connected to the BBN LLE, giving the appearance to the end systems that they were directly connected over a single OC-3 link, with no intervening switches. This setup is depicted in Figure 3.

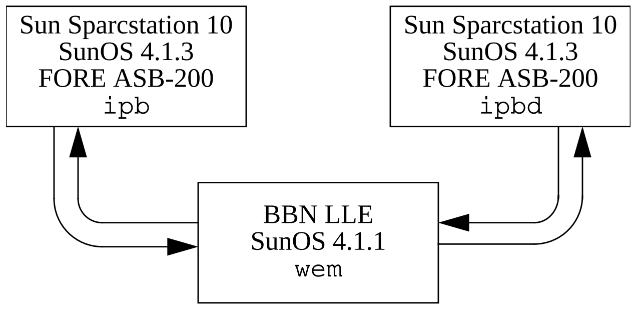

FIGURE 3. Experimental Setup. Arrows show the direction of data flow. Connections to production Ethernet and FDDI networks are not shown.

We established a pair of Permanent Virtual Circuits (PVCs) to carry our test data over the link. We also configured the drivers and routing tables on the Sparcstations to forward all IP packets between them over our PVCs. The IP Maximum Transmission Unit (MTU) over the IP-over-ATM link was the default value of 9188 bytes. Finally, we used FDDI interfaces on the two Sparcstations and the Ethernet interface on the LLE for remote logins and control functions.

4 Experimental Results

In this section, we present some initial measurements we performed using the BBN LLE and the `ipb` and `ipbd` benchmark programs. The results we obtained are summarized in Table 2, and discussed in some detail in the remainder of this section.

The mean HTTP file transfer can take as much as seven times longer to complete with the introduction of a link with a 10^{-6} bit error rate. The mean Web page transfer can take four times longer to complete.
Long links tend to cluster HTTP page and file retrieval times around integer multiples of the round-trip time. This effect gradually fades away for long transfers on a lossy link, due to the effects of TCP timeouts and retransmissions.
Our simulated error-prone link shows a large effect on the 90th- and higher-percentile file and page transfer times; it has little effect on the median transfer times.
The effects of errors are proportionately larger for short-delay links, although the absolute difference in transfer times is more for long-delay links.

TABLE 2. Summary of Simulation Results.

4.1 Delay Effects

We used the LLE to vary the one-way propagation delay on the link between the two Sparcstations in the range of 1–200 ms. We set identical delays on the two sides of the bidirectional link, although the hardware would have allowed us to configure asymmetric delays (such a setup could occur in a network using a satellite link for distribution and a terrestrial modem backchannel). For this first set of runs we disabled the LLE's error generator. Aside from the traffic generated by our experiments, the link was idle.

We performed three measurement runs of the benchmark at each of four delay settings, with new clients entering the system every 15 seconds on average. Each run lasted approximately an hour. Table 3 shows statistics describing the response times for all of the files retrieved at each delay setting (aggregated across all runs). Each delay time in these sample distributions indicates the amount of time elapsed between the sending of a request and the receipt of the last byte of the reply, corresponding to a single HTTP file transaction.

Figure 4 depicts the Cumulative Distribution Functions (CDFs) of the retrieval times, for different propagation delays. As can be seen, the file retrieval times are highly influenced by long RTTs; there are pronounced "steps" at integer multiples of the round-trip times, especially visible with 100 ms and 200 ms propagation delays. We note that with such long RTTs, the delay-bandwidth

Delay	1 ms	10 ms	100 ms	200 ms
Number	5733	5693	5314	5130
Mean	16.4 ms	52.9 ms	535.1 ms	927.9 ms
Min	3.5 ms	21.5 ms	201.4 ms	401.4 ms
Max	624.9 ms	929.0 ms	38827.6 ms	41959.7 ms
Median	8.3 ms	25.4 ms	205.0 ms	406.1 ms

TABLE 3. HTTP File Retrieval Times on Error-Free Link. Columns correspond to varying one-way propagation delays.

product can be considerable (8 MB in the case of a 200 ms propagation delay, leading to a 400 ms RTT). By contrast, the median Web file size in the model of [8] is approximately 2 KB. For these short files, the TCP slowstart algorithm dictates that the server sends small segments of the file and awaits acknowledgments from the client, gradually building up the TCP congestion window. This send-acknowledge style of communication persists until the TCP congestion window opens sufficiently to allow steady-state streaming of data; small HTTP file transfers do not involve enough packets to permit this to happen (note that 80% of file transfers completed in 2 RTTs or less).

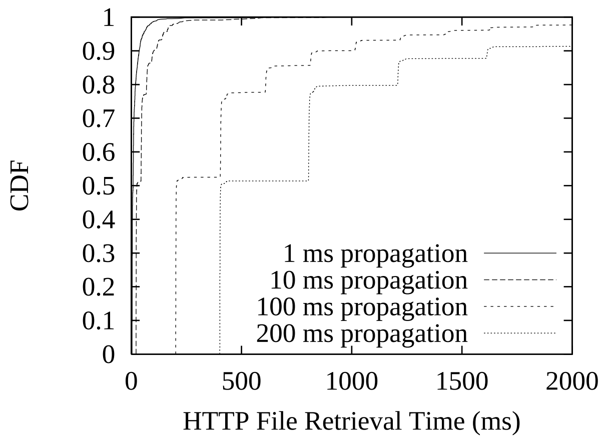

FIGURE 4. Cumulative Distribution Function of HTTP File Retrieval Times, Error-Free Link.

Web pages are composed of one or more files. In Table 4, we present the statistics of the page retrieval times, with varying propagation times. Each of the page retrieval times measures the elapsed time between the sending of the first request on a given Web page and the receipt of the last byte of the last reply on that page. It is a measure of how long a user needs to wait for the downloading of a complete Web page. We note again that, due to the fact that there may be multiple outstanding requests for a given Web page, the total retrieval time for a page may be less than the sum of the retrieval times of each of its component files.

Delay	1 ms	10 ms	100 ms	200 ms
Number	2105	2100	1986	1880
Mean	14.4 ms	55.2 ms	548.1 ms	975.7 ms
Min	3.5 ms	21.5 ms	201.5 ms	401.4 ms
Max	629.0 ms	951.0 ms	39029.5 ms	41959.7 ms
Median	8.2 ms	44.3 ms	404.3 ms	804.2 ms

TABLE 4. HTTP Page Retrieval Times on Error-Free Link. Columns correspond to varying one-way propagation delays.

We present the CDFs of the retrieval times of complete Web pages in Figure 5. As would be expected, the page retrieval times tend to be longer than those for individual pages. However, the CDFs still exhibit a similar "stair-step" shape, indicating that the page retrieval times tend to cluster around some multiple of the RTT, for high RTTs. This fact is not terribly surprising, given that most Web pages consist of small number of files (which are, themselves, short).

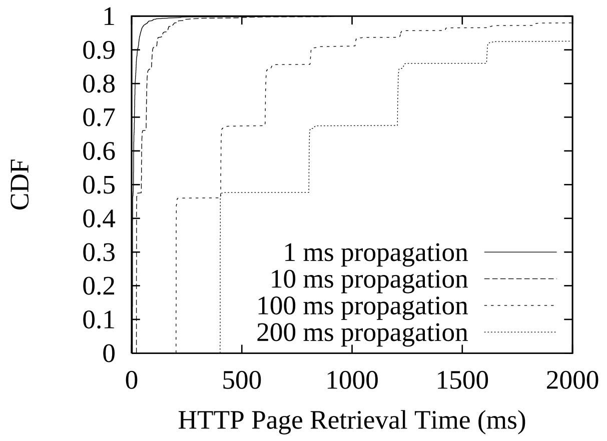

FIGURE 5. Cumulative Distribution Function of HTTP Page Retrieval Times, Error-Free Link.

4.2 Error Effects

We next ran the same experiments with the LLE providing a lossy ATM link, such as might be found in a wireless or satellite ATM environment. We programmed the LLE to provide isolated bit errors, with a BER of 2^{-20}, or approximately 10^{-6}. Tests using the UNIX `ping`(1) utility showed that this BER translates to a packet loss rate of approximately 1% for 1 KB packets. Other than the use of the LLE's error generators, our setup was the same as in Section 4.1.

Table 5 shows some of the statistics on simulated performance with the wireless emulator active. As expected, the mean response times are several times longer than those obtained with an error-free link. However, the median response times were largely unaffected, likely due to the discretization of response times to integer multiples of the RTTs.

Delay	1 ms	10 ms	100 ms	200 ms
Number	5520	5929	4808	4342
Mean	114.3 ms	152.4 ms	655.0 ms	1088.0 ms
Min	3.5 ms	21.5 ms	201.5 ms	401.4 ms
Max	10845.1 ms	9533.8 ms	31836.3 ms	52017.8 ms
Median	8.1 ms	26.9 ms	207.9 ms	405.3 ms

TABLE 5. HTTP File Retrieval Times on a Lossy ATM Link. Columns correspond to varying one-way propagation delays.

This effect is quite visible in Figure 6, which shows the CDF of the file retrieval times with a lossy link. In both the error-free and lossy cases, the first "plateau" in the CDF, coming between the first two modes, remains just above the median. As the modes of these distribution are still determined by the RTT, the median file response time is nearly the same as with the error-free link. Qualitatively, Figure 6 also shows that at long retrieval times, the introduction of errors smooths out the clustering effects around integer multiples of the RTT. This effect, especially noticeable above the 90th percentile, is likely caused by TCP timeouts and retransmissions. The TCP retransmit timers have a fixed granularity of 500 ms in most implementations, which bears no relation to the RTT. Moreover, they disrupt the pattern of data transmission, as the TCP congestion window closes after each timeout-based retransmission.

More quantitatively, we observe that the higher percentiles of file transfers were affected more than lower percentiles by the presence of errors on the ATM link. We show these effects for two different propagation delay values of 1 ms and 200 ms, in Table 7. This tendency was generally present for all four propagation delay values we tested, in the individual datasets as well as the aggregations presented here. This result is expected if we note that long files, presumably accounting for long transfer times, are more likely to suffer a link error while they are being sent.

We noted an interesting effect in cases of very long file transfer times, above the 95th percentile. In this regime, the LLE-induced errors had proportionately more effect across short-delay hops than across long-delay hops. This trend was generally consistent across the individual

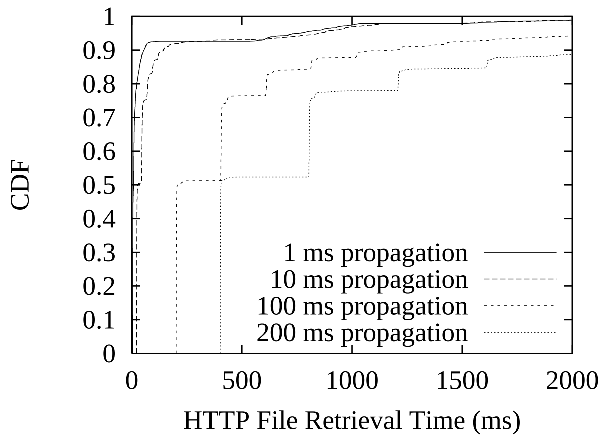

FIGURE 6. Cumulative Distribution Function of HTTP File Retrieval Times, Lossy Link.

Percentile	Error-Free	Wireless	% Increase
70	12.2 ms	12.4 ms	1.6%
80	19.5 ms	22.8 ms	16.9%
90	35.8 ms	55.2 ms	54.2%
95	54.4 ms	769.2 ms	1314.0%
99	117.0 ms	2064.8 ms	1664.8%

TABLE 6. Comparison of File Response Times at 1 ms One-Way Propagation Delay.

runs, as well as the aggregation of all runs, although there were some departures at 100 ms. We show this information in Table 7. We believe that this effect is caused by the TCP retransmit timers; in case of a timeout-based retransmission, the 500 ms granularity will have a proportionately greater effect on transfers across a short link because it is a larger percentage of the link propagation time.

%-ile	1 ms	10 ms	100 ms	200 ms
70	1.6%	0.6%	0.0%	-0.0%
80	16.9%	3.0%	0.3%	0.1%
90	54.2%	28.5%	25.1%	24.8%
95	1314.0%	389.0%	51.2%	27.8%
99	1664.8%	621.3%	69.8%	84.5%

TABLE 7. Percentage Increase in Transfer Times Incurred by Lossy Link.

The effects of the lossy link on Web page retrieval times were quite similar to those for individual Web files. We present the former statistics, with the LLE in operation, in Table 8. Compared to the error-free trials in Table 4, the mean retrieval times increased from 1.1 to 6.4 times.

Delay	1 ms	10 ms	100 ms	200 ms
Number	2036	2231	1804	1654
Mean	92.7 ms	127.6 ms	644.2 ms	1099.6 ms
Min	3.5 ms	21.5 ms	201.5 ms	401.4 ms
Max	10845.1 ms	9555.6 ms	31836.3 ms	52420.2 ms
Median	8.4 ms	44.3 ms	404.4 ms	803.8 ms

TABLE 8. HTTP Page Retrieval Times on a Lossy ATM Link. Columns correspond to varying one-way propagation delays.

Figure 7 pictures the CDF of page retrieval times. As with file retrievals, the use of a lossy link had the effect of spreading out the clustering of page response times from around integer multiples of the RTT. This effect is visible in the gradual disappearance of the “steps” in the CDF, particularly noticeable with a 100 ms propagation delay, as the retrieval time increases.

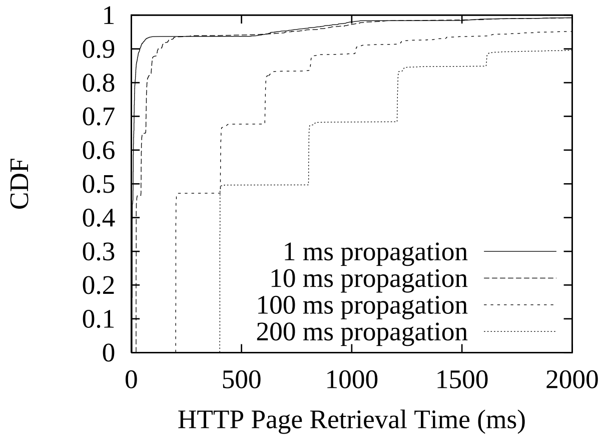

FIGURE 7. Cumulative Distribution Function of HTTP Page Retrieval Times, Lossy Link.

As with file retrievals using the emulated wireless link, errors had a more pronounced effect on high-percentile transfer times. For high percentiles, the effects of errors were again more prominent on short-delay links than on long-delay links.

5 Summary and Future Work

We have constructed the IP Benchmark (IPB), a tool to investigate the performance of IP networks under the simulated traffic loads of different applications. Our approach relies on mimicking the network traffic patterns of applications such as World Wide Web clients and servers, suitably regulated by transport protocol implementations. We measure the performance received from the network and report these findings in terms of application-specific metrics.

We have begun some measurements using this setup to measure the performance of simulated World Wide Web traffic. Our initial results show that over an emulated lossy ATM link (with a 10^{-6} bit error rate), the mean Web file takes as much as seven times longer to transfer, with the mean Web page taking up to four times longer to be received. We observe the quantization of transfer times around integer multiples of the link's round-trip time. The effects of the lossy link seemed to be most evident for the longest transfers. Finally, we saw that the simulated wireless link had (proportionately) the largest effect when the link had a short round-trip time.

There are a number of areas for future work. The incorporation of more traffic types (such as CBR audio and various types of network video), as well as updated models of existing applications, would improve the utility of `ipb` as a tool to predict network performance. With respect to our measurements of emulated wireless and satellite networks, it would be useful to be able to "slow down" the wireless link from its current OC-3 (155 Mbps) rate to more realistic WATM radio speeds (typically tens of megabits per second). Finally, incorporating a range of different WATM network types into our wireless link emulator would allow some comparisons between different link types.

6 References

[1] The Apache Group. *Apache software*, 1995–1997. This software is available at http://www.apache.org.

[2] Tim Berners-Lee and Daniel W. Connolly. *Hypertext Markup Language – 2.0*. Internet Request for Comments 1866, November 1995.

[3] Tim Berners-Lee, Roy T. Fielding, and Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0*. Internet Request for Comments 1945, May 1996.

[4] Peter B. Danzig and Sugih Jamin. tcplib: A library of TCP internetwork traffic characteristics. Technical Report USC-CS-91-495, Computer Science Department, University of Southern California, Los Angeles, CA, 1991.

[5] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. Internet Request for Comments 2068, January 1997.

[6] Bruce A. Mah. *INSANE Users Manual*. Computer Science Division, University of California at Berkeley, May 1996.

[7] Bruce A. Mah. *Quality of Service and Asynchronous Transfer Mode in IP Internetworks*. PhD dissertation, University of California at Berkeley, December 1996.

[8] Bruce A. Mah. An empirical model of HTTP network traffic. In *Proceedings of IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[9] Walter Millikan. *The Long Link Emulator: System Description*. BBN Systems and Technologies, September 1993.

[10] Netscape Communications Corporation. *Netscape Navigator software*, 1994–1998. This software is available at http://home.netscape.com/.

[11] Giao T. Nguyen, Randy H. Katz, Brian Noble, and Mahadev Satyanarayanan. A trace-based approach for modeling wireless channel behavior. In *Proceedings of the Winter Simulation Conference*, December 1996.

[12] Jon Postel. *Transmission Control Protocol*. Internet Request for Comments 793, September 1981.

[13] Jon Postel and Joyce Reynolds. *Telnet Protocol Specification*. Internet Request for Comments 854, May 1983.

[14] *Squid Internet Object Cache*, 1997. This software is available at http://squid.nlanr.net/Squid/.

[15] W. Richard Stevens. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. Internet Request for Comments 2001, January 1997.

[16] *ttcp*, 1991. This software is available at ftp://ftp.sgi.com/sgi/src/ttcp/.